Defining a Process for Simulation Software Vulnerability Assessments

Dr. John A. Hamilton Jr.
Auburn University

Col. Kevin J. Greaney
Missile Defense Agency

Gordon Evans
Booz Allen Hamilton

The need for simulation software vulnerability assessment is being driven by three major trends. They are increased use of modeling and simulation for training and operational planning, increased emphasis on coalition warfare and interoperability, and increased awareness of the potential security risks inherent in sharing operationally useful software. This article will describe in an unclassified manner the process developed by the U.S. Missile Defense Agency and Auburn University to evaluate potential vulnerabilities in shared simulation software.

This may seem an obvious point to CrossTalk readers, but many members of the Department of Defense modeling and simulation community are not software engineers. Therefore, the model's software implementation must be analyzed as well as the model itself.

The security model for many missile defense simulations is similar to that used in the heyday of the U.S. Army's nuclear weapons training. When virtually all U.S. Army medium and heavy artillery batteries were nuclear capable, it was necessary to conduct training for units, particularly Reserve Component units that did not have secure facilities to handle classified models. The result was an unclassified training system that only provided classified results when given actual weapons performance data. Soldiers were thus able to train on how to do the targeting calculations without handling classified information as shown in Figure 1.

Applying this model to missile defense simulations is more difficult because the calculations are much more complex, and there are many more parameters to deal with. Furthermore, these simulations are implemented in software, and that increases the complexity. Therefore, the Missile Defense Agency's Models and Simulation Directorate (MDA/SES) developed a vulnerability assessment process in partnership with Auburn University's Information Assurance Laboratory. We next describe the process and the environment that framed our process.

Assessing the Threat

U.S. missile defense programs, training, tactics, and procedures are a matter of intense interest to foreign intelligence agencies. Intelligence agencies do not face the same economic constraints, as do practitioners of economic espionage. For this reason, military-relevant software may be attacked in ways that would not be feasible for an industrial reverse-engineering application.

An unclassified analysis of one missile defense simulation Web site showed that nearly one third of the site's hits could be traced back to the People's Republic of China [1]. The largest number of recorded hits came from Beijing, and this was more than twice the number of hits from any other country, including the United States. This obviously does not include undetected intrusions.

> *"An unclassified analysis of one missile defense simulation Web site showed that nearly one third of the site's hits could be traced back to the People's Republic of China."*

Defining a Process

Information compiled into binary code is not secure. Just because it is hard to extract information from a binary file does not mean it is impossible to do so [2]. As shown in Figure 2, our process analyzes inputs, outputs to the simulation software, and the executable binaries.

Phase 1: Inputs

When analyzing the system inputs, we look at how well we can simulate a system based on open-source data. Next, we search for buffer overflows. Given the popularity of the C programming language, it is usually not hard to find a buffer overflow – either in the application or in the operating system it is running on. Whether a buffer overflow can be used to compromise sensitive information in the application remains to be seen. Theoretically, one could jump to a code segment written to start dumping out intermediate calculations. Operating systems are written in C and are vulnerable to buffer overflows, too.

Buffer overflows can be used for more than just jumping a program to an unauthorized code segment. We found that entering control characters into an entry screen would bring up a debugger providing important clues on how the program was originally compiled.

What if the simulation developers used explicit bounds checking for every input? One thing to look for is any sensitive information that can be gleaned from the bounds. An interceptor that has actual minimum and maximum ranges as bounds would be an example of a possible vulnerability.

Phase 2: Attacking the System

Simulation software runs on top of operating systems. On distributed simulation implementations, operating system vulnerabilities may be exploited to remotely compromise the simulation software. It is often instructive to study the installed files of a software distribu-

Figure 1: *U.S. Army Nuclear Weapons Training Model Circa 1980 (Unclassified)*

tion to learn more about the program structure and contents.

A good definition for reverse engineering can be found at [3]. Van Deursen defines reverse engineering as,

> The process of analyzing a subject system with two goals in mind: (1) to identify the system's components and their interrelationships, and (2) to create representations of the system in another form or at a higher level of abstraction. [3]

In this process, we look at both disassembly of code as well as decompilation.

Disassembly is reconstructing assembly language code from a binary. Eric Imsand and Adam Sachitano have disassembled missile defense simulations using *dis* on Solaris and a shareware program called Hackman on Windows platforms. Both dis and Hackman are disassembly programs. They report the following:

> The Hackman application ran easily. After launching the program, it was simply a matter of selecting the tool, either a hex editor or disassembler, and choosing the file to open. *Hackman* opened the file, and disassembled it without further user interaction. The entire disassembly process took approximately six hours running on a 400 MHz Intel Celeron processor with 128 MB of RAM. [4]

Imsand and Sachitano produced about one gigabyte of assembly code and were later able to reassemble the binary and successfully run it.

With assembly code in hand, it is possible to insert additional instructions to create a modified binary that dumps every variable value to an output device. It is also possible to search for string literals.

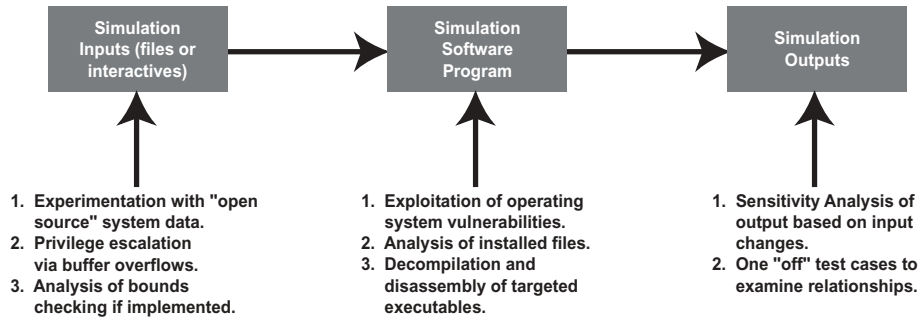

Figure 2: *Process for Simulation Software Vulnerability Analysis*

Decompilation is the generation of high-level source code from low-level input [5]. We have experienced little success in decompiling, primarily due to our reliance on freeware and shareware tools. Commercial decompilers are available and the state of the art in this area continues to improve.

Weide, Heym, and Hollingsworth discuss reverse engineering of large legacy software systems. They conclude that the reverse engineering of such systems is *intractable* in the sense that if one is given real (high-level) legacy code, the time required to show the validity of an explanation for why it exhibits a certain behavior is at least exponential compared to the size of the source code [6]. However, their same paper asserts a caveat that is repeated here: "This does not mean that the task is impossible. It means that it is prohibitively costly for large systems" [6]. We would add that what is prohibitively expensive in the commercial sector is not necessarily prohibitively expensive for a high-priority intelligence effort.

Phase 3: Outputs

We attempt to determine the internals of the programs by analyzing the outputs and their sensitivity to changes based on carefully chosen inputs. In general, we believe that missile defense simulations of any importance are too complicated to make this a useful strategy for reverse engineering the simulation. However, it is possible to gain insight into specific aspects of the simulation by constantly running it and making minor changes to the input and tracking the changes. These one *off* test cases are constructed by varying only one input parameter. If the simulation is well documented, this strategy can be used in conjunction with an analysis of the documentation to determine internal relationships between parameters.

Refining and Applying the Process for Different Levels of Assurance

Given the costs associated with vulnerability analysis, we defined three sets of tasks providing three levels of assurance: High, Medium, and Low. These categories reflect the level of effort required for the analysis. The requirements for each are enumerated in Table 1.

It is important to recognize that anything sensitive in the source code is vulnerable. It is hard, time consuming, and expensive to get at it – but it is naïve to think that a hostile intelligence agency would not make such an attempt. Next, we address each item in Table 1.

- **Source Code.** A line-by-line verifica-

Table 1: *Assurance Levels for Simulation Software Vulnerability Analysis*

High Assurance Level	Medium Assurance Level	Low Assurance Level
Line-by-line verification of source code.	Line-by-line verification of selected source code.	Line-by-line verification of selected source code.
Professional decompilation of executables.	String search on disassembled code.	String search on disassembled code.
Complete review of published documentation.	Targeted review of published documentation.	Targeted review of published documentation.
Open source review of weapons and systems data.	Open source review of weapons and systems data.	Analysis of degree of parameterization.
Analysis of simulation runs to evaluate training, tactics, and procedures.	Analysis of simulation runs to evaluate training, tactics, and procedures.	
Analysis of degree of parameterization.	Analysis of degree of parameterization.	

tion of a simulation with a million+ lines of code is nontrivial. Worse, there is no guarantee that such a massive effort will uncover all potential security issues. However, it is the best way to detect problems. Software engineering research has long held that the best way to find any problem in software is through desk-checking the source code. In most cases, a line-by-line verification will not be warranted. If (and this is a big if) the simulation software is well structured, then it is reasonable to exclude large portions of the code and simply focus on the modules that deal with sensitive issues. It can be argued that a more focused review of *high-risk* code could potentially be more fruitful than plowing through a massive program in its entirety. Any source code provided, as part of the distribution must be reviewed. Source code analysis can give you a worst-case vulnerability assessment.

- **Decompilation/Disassembly.** Decompilation and disassembly can be used to provide an expected case analysis. We pursue this to see what a potential adversary can learn from the binaries. For high assurance requirements, we recommend using professionals to decompile the binaries. Open market decompilers (available to a university anyway) are not yet to a point where experienced software engineers can gain useful results through reasonable efforts. We have no insight into what tools are available in the world of restricted access programs, but we believe that much better tools are theoretically possible and practical. Dissemblers are readily available and useful. It is reasonable to write scripts to do string searches on massive assembly code files and prudent to do that. In all cases, the binaries should be checked to make sure that all debugging information is stripped before the binaries are released.
- **Documentation Review.** Documentation of simulations must be included in the distribution [7]. Some simulations include more than 1,000 pages of documentation. Documentation is critical to the successful utilization of a simulation. As Sargent notes:

> Documentation on model verification and validation is usually critical in convincing users of the 'correctness' of a model and its results, and should be included in the simulation model documentation. [8]

The caveat to Sargent's assertion is that the documentation must be reviewed to make sure that no sensitive information is inadvertently released. The physics of missile trajectories are not sensitive; probability of kill for a given system is very sensitive.

- **Open Source Review.** There is a great deal of published information on missile and missile defense systems, particularly older ballistic missile systems such as scuds. One way to exercise a simulation is to create models from open source material and then experiment with them.

"It is important to recognize that anything sensitive in the source code is vulnerable. It is hard, time consuming, and expensive to get at it – but it is naïve to think that a hostile intelligence agency would not make such an attempt."

- **Analysis of Simulation Runs.** Using open source inputs provides the means to develop simulation runs and analyze the outputs. The objective is to reduce the number of unknowns in the system. The more known information that can be input, the easier the analysis.
- **Analysis of Degree of Parameterization**. Essentially, we want to verify that the model is unclassified and that classified results are only produced when classified parameters are used. If there are default values, then those values need to be checked to see if any are sensitive in nature. In general, the greater the degree of parameterization, the closer the simulation approximates the model in Figure 1.

Conclusion

In most cases, we believe that a medium assurance assessment is sufficient. Before we share simulations (missile defense or others) with our coalition partners, it is essential to know what we are sharing.

This research has demonstrated a viable, scaleable means of assessing the vulnerability of complex simulation software. We believe this methodology is appropriate for use with other simulation programs. It is always difficult to prove a negative. We do not claim that our process can prove the absence of vulnerabilities or find every vulnerability in every software implementation. However, this process can provide an important means of risk mitigation. We believe the process defined here can successfully identify vulnerabilities in simulation software.◆

References

1. Mann, Steve. "Default Report Web Site Visitors." WebTrends. NetIQ Corporation, Internal Report. 17 May 2002: 41.
2. Viega, John, and Gary McGraw. Building Secure Software. Boston, MA: Addison-Wesley, 2002: 109.
3. Van Deursen, Arie. "Reverse Engineering." Jan. 2003 <www.program-transformation.org/twiki/bin/view/Transform/ReverseEngineering>.
4. Imsand, Eric, and Adam Sachitano. "Analyzing Security Vulnerabilities in National Missile Defense Simulation Software." Unpublished, Nov. 2002.
5. Breuer, Peter T., and Jonathan P. Bowen. "Decompilation: The Enumeration of Types and Grammars." ACM Transactions on Programming Languages and Systems 16. 5 (1994).
6. Weide, Bruce W., Wayne D. Heym, and Joseph E. Hollingsworth. Reverse Engineering of Legacy Code Exposed. Proc. of the 17th International Conference on Software Engineering, Seattle, WA. New York: ACM Press, 1995: 327-331.
7. Chatham, Wade. "A Vulnerability Analysis with an Emphasis on Using Documentation." Unpublished, Nov. 2002.
8. Sargent, R. "Verifying and Validating Simulation Models." Proc. of the 28th Winter Simulation Conference, Coronado, CA. New York: ACM Press, 1995: 55-64.

About the Authors

John A. "Drew" Hamilton Jr., Ph.D., is an associate professor of computer science and software engineering at Auburn University and director of Auburn University's Information Assurance Laboratory. Prior to his retirement from the U.S. Army, he served as the first director of the Joint Forces Program Office and on the staff and faculty of the U.S. Military Academy, as well as chief of the Ada Joint Program Office. He has a Bachelor of Arts in journalism from Texas Tech University, masters degrees in systems management from the University of Southern California and in computer science from Vanderbilt University, and a doctorate in computer science from Texas A&M University.

Auburn University
107 Dunstan Hall
Auburn, AL 36849
Phone: (334) 844-6360
Fax: (334) 844-6329
E-mail: hamilton@eng.auburn.edu

Col. Kevin J. Greaney, U.S. Army, is the director, Models and Simulations, Missile Defense Agency. Prior to his assignment, Greaney served as the commander of the Communication Electronics Command Software Engineering Center-Meade from September 1997 through September 2000. Greaney was selected as a Distinguished Military Graduate prior to his commission as a second lieutenant. He has a Bachelor of Arts from Northeastern University, a Master of Science from Shippensburg University, a Master of Arts from Webster University, and is currently pursuing a doctorate in software engineering from the Naval Post Graduate School.

Missile Defense Agency
Pentagon
Washington, D.C.
Phone: (703) 697-4360
Fax: (703) 695-6133
E-mail: kevin.greaney@bmdo.osd.mil

Gordon Evans is a consultant for Booz Allen Hamilton working on-site at the Missile Defense Agency (MDA). His areas of concentration have been in systems engineering, command and control, modeling and simulations, international programs, and technology transfers. He has been the lead MDA designer and investigator for its Modeling & Simulation Vulnerability Assessment program. Evans retired from the U.S. Army in 1992 as a lieutenant colonel. During his military service, he served in multiple Field Artillery and Military Intelligence assignments. Overseas assignments have been in Germany, Korea, and Vietnam.

Booz Allen Hamilton – MDA/SES
Pentagon
Washington, DC
Phone: (703) 697-4582
Fax: (703) 695-6133
E-mail: gordon.evans-contractor @ bmdo.osd.mil